UNIKASSEL VERSITÄT

SOFTWARE ENGINEERING
UNIVERSITÄT KASSEL

Ein *Simple Genetic Algorithm* (SGA)

Jens Kosiol

(Folien basieren auf Kap. 2 des Buchs von Michalewicz)

Ein *Simple Genetic Algorithm* (SGA)

sga(p_c, p_m, f, n, t, m)
  $P \leftarrow$ *initialize population(m,n); //* P Multimenge
  while t do
    $Q \leftarrow$ *selectParents(P,f);*
    $Q \leftarrow$ *applyCrossover(*Q*,*p_c*);*
    $P \leftarrow$ *applyMutation(*Q, p_m*);*
    update t;
  end
  return P;

Input

p_c: Wahrscheinlichkeit für Crossover
p_m: Wahrscheinlichkeit für Mutation
f: Fitnessfunktion (inklusive Dekodierung)
n: Populationsgröße
t: Terminationsbedingung
m: Benötigte Länge der Kodierung

Im Folgenden: Durchgang der einzelnen Funktionen für Bitvektoren als Kodierung

Genotyp vs. Phänotyp

Genotyp

- Kodierung der potentiellen Lösungen
- Hierauf arbeiten die evolutionären Operatoren
- Beispiele:
  - Vektoren/Strings von Bits, natürlichen oder reellen Zahlen
  - Permutationen
  - Graphen
  - …

Phänotyp

- Die eigentlichen potentiellen Lösungen
- Hierauf wird die Fitness ausgewertet
- Beispiele:
  - (Vektoren von) natürliche(n) oder reelle(n) Zahl(en)
  - Position von Damen auf einem Schachbrett
  - Untergraph eines gegebenen Graphen
  - Eine Spielstrategie
  - …

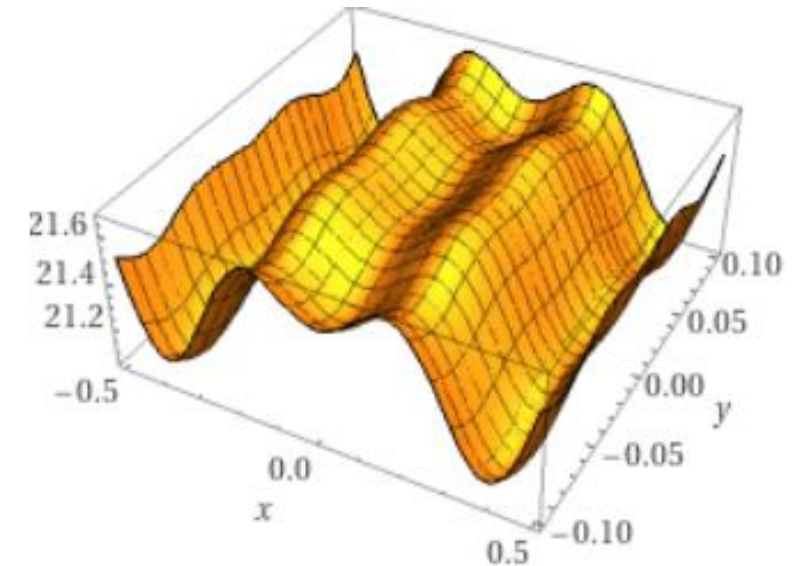
Ein Beispielproblem

Maximiere die Funktion

$$g = 21{,}5 + x\sin(4\pi x) + y\sin(20\pi y)$$

auf dem Gebiet $[-3, 12.1] \times [4.1, 5.8]$

- Kein globales Maximum auf $\mathbb{R} \times \mathbb{R}$
- $x\sin(4\pi x)$ hat lokale Maxima bei $x \approx 11.6171$ bzw. $x \approx 11.6369$ mit Wert ≈ 11.5078 …
- $y\sin(20\pi y)$ hat lokales Maximum bei $y \approx 5.72504$ mit Wert 5.72502 …
- Also hat g ein lokales Maximum bei $\approx (11.6171, 5.72504)$ mit Wert ≈ 38.73282



Kodierung als Bitvektor/Bitstring

- Kodierung reeller Zahlen oft als Gleitkommazahl
- Alternative bei fixem Intervall: Aufteilung des Intervalls in Segmente (je nach gewünschter Genauigkeit) und (binäre) Durchnummerierung
- Intervall [-3, 12.1] auf 4 Nachkommstellen Genauigkeit:

$$2^{17} < 151000 < 2^{18} - 1$$

- Für allgemeines Intervall $[a, b]$: Bestimme m, sodass

$$2^{m-1} < (b-a) * 10^{l} \leq 2^{m} - 1,$$

wobei l die gewünschte Anzahl an Nachkommastellen ist

Kodierung als Bitvektor/Bitstring II

Sei $[a, b]$ das Intervall, das binär kodiert werden soll, und m die benötigte Länge des Bitvektors, die sich aus der gewünschten Genauigkeit ergibt.

- Dekodierung eines Bitvektors v zu reeller Zahl:

$$\text{decode}(v) = a + v_{\text{Dez}} \times \frac{b-a}{2^m-1},$$

wobei $(_)_{\text{Dez}}$ die Umrechnung ins Dezimalsystem bezeichne.

- Kodierung einer reellen Zahl $r \in [a, b]$ zu einem Bitvektor:

$$\text{encode}(r) = \left(\frac{r-a}{b-a} \times (2^m - 1)\right)_{\text{Bin}},$$

wobei $(_)_{\text{Bin}}$ die Umrechnung ins Binärsystem bezeichne und r auf die gewünschte Genauigkeit gerundet ist. Die berechnete Binärzahl wird dann (von links) mit Nullen auf die Länge m aufgefüllt.

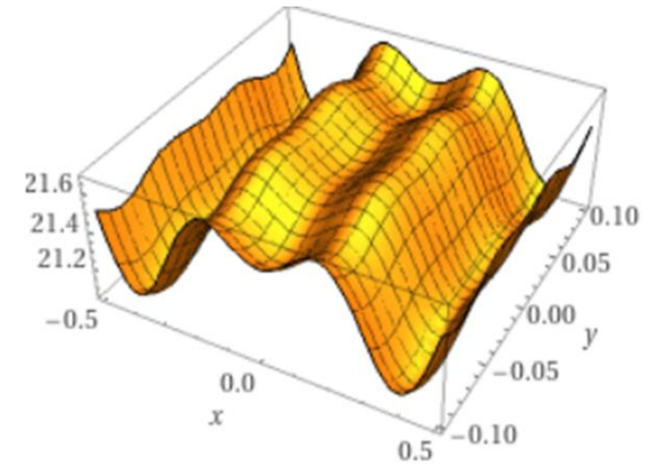
- Bei mehreren Variablen entsteht die Gesamtkodierung durch Konkatenation der einzelnen Kodierungen.

Beispiel Kodierung

Maximiere die Funktion

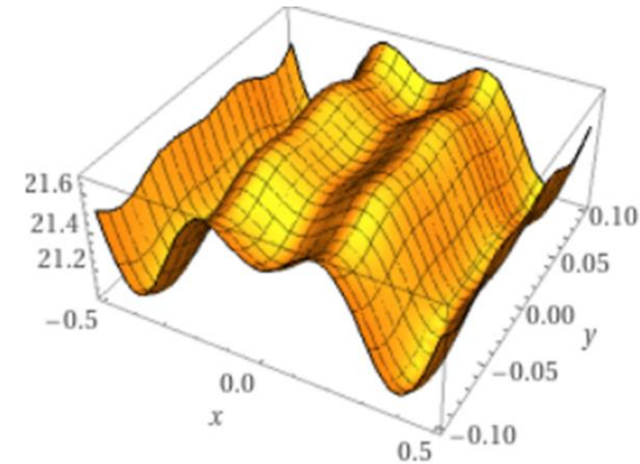
$$g = 21{,}5 + x\sin(4\pi x) + y\sin(20\pi y)$$

auf dem Gebiet $[-3, 12.1] \times [4.1, 5.8]$.



- Lösungsraum sind also Tupel reeller Zahlen $(r_1, r_2) \in [-3,\ 12.1] \times [4.1, 5.8]$.
- Suchraum sind Bitstrings der Länge 33; r_1 kodiert zu den ersten 18 Bits und r_2 zu den folgenden 15.
- Im Verlauf eines evolutionären Algorithmus müssen wir normalerweise keine Lösungen kodieren, sondern nur Chromosomen dekodieren.

Beispiel Dekodierung



Wir betrachten das Chromosom

$$v = (010001001011010000111110010100010)$$

als Kodierung eines Tupels reeller Zahlen
$(r_1, r_2) \in [-3,\ 12.1] \times [4.1, 5.8]$.

- Die ersten 18 Ziffern kodieren r_1 und die nächsten 15 Ziffern kodieren r_2.
- $\text{decode}(010001001011010000) =$
  $-3 + (010001001011010000)_{\text{Dez}} \times \frac{12.1-(-3)}{2^{18}-1} = 1.0524$
- $\text{decode}(111110010100010) =$
  $4.1 + (111110010100010)_{\text{Dez}} \times \frac{5.8-4.1}{2^{15}-1} = 5.7553$
- Insgesamt dekodiert v zu $(1.0524, 5.7553)$

Ein *Simple Genetic Algorithm* (SGA)

sga(p_c, p_m, f, n, t, m)
  $P \leftarrow$ *initialize population(m,n);* // P Multimenge
  while t do
    $Q \leftarrow$ *selectParents(P,f);*
    $Q \leftarrow$ *applyCrossover(Q,*p_c*);*
    $P \leftarrow$ *applyMutation(*Q, p_m*);*
    update t;
  end
  return P;

Input

p_c: Wahrscheinlichkeit für Crossover
p_m: Wahrscheinlichkeit für Mutation
f: Fitnessfunktion (inklusive Dekodierung)
n: Populationsgröße
t: Terminationsbedingung
m: Benötigte Länge der Kodierung

SGA – Initialisierung von Bitvektoren

Initialisierung hängt ab von gewählter Populationsgröße n und benötigter Länge der Kodierung m: Erzeuge n zufällige Bitvektoren der Länge m.

Eine Möglichkeit der Umsetzung: Wert jedes Bits ist unabhängig und gleichverteilt 0 oder 1; also: An jeder Position jedes Vektors wird mit Wahrscheinlichkeit $0{,}5$ eine 0 eingetragen und mit Wahrscheinlichkeit $0{,}5$ eine 1.

Implementierung: Erzeuge zufällig gleichverteilt eine reelle Zahl $r \in [0{,}1]$. Falls $r \leq 0{,}5$, nimmt das nächste Bit den Wert 0 an, sonst den Wert 1.

Ein *Simple Genetic Algorithm* (SGA)

sga(p_c, p_m, f, n, t, m)
  $P \leftarrow$ *initialize population(m,n);* // P Multimenge
  while t do
    $Q \leftarrow$ *selectParents(P,f);*
    $Q \leftarrow$ *applyCrossover(Q,*p_c*);*
    $P \leftarrow$ *applyMutation(*Q, p_m*);*
    update t;
  end
  return P;

Input

p_c: Wahrscheinlichkeit für Crossover
p_m: Wahrscheinlichkeit für Mutation
f: Fitnessfunktion (inklusive Dekodierung)
n: Populationsgröße
t: Terminationsbedingung
m: Benötigte Länge der Kodierung

Fitnessfunktion

Die Fitnessfunktion $f: S' \rightarrow \mathbb{R}$ misst die Qualität eines Lösungskandidaten. Hierbei ist S' die Menge aller unkodierten Lösungen (Phänotypen).

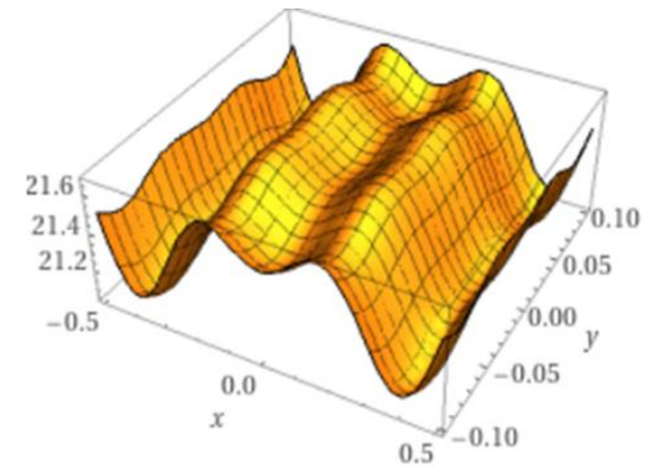
Auswertung der Fitness eines Chromosoms v geschieht per $f(\text{decode}(v))$, wobei S der Suchraum ist und $\text{decode}: S \rightarrow S'$ Chromosomen dekodiert.

Je nach Art des Optimierungsproblems bedeutet für Lösungen s, s', falls $f(s) \leq f(s')$, dass s die bessere Lösung ist (Minimierungsproblem) bzw. die schlechtere (Maximierungsproblem).

Spezialfall: Fitness ist keine Funktion sondern (partielle) Ordnung: Wir kennen keine absoluten Fitnesswerte von Lösungen, sondern können zwei Lösungen nur darauf vergleichen, welche besser ist.

Beispiel Fitnessfunktion

- Als Fitness $f(x_0, y_0)$ wählen wir den Wert der Funktion $g = 21{,}5 + x\sin(4\pi x) + y\sin(20\pi y)$ an der Stelle (x_0, y_0).
- Da wir ein Maximum suchen, ist (x_0, y_0) "fitter" als (x_1, y_1), falls $f(x_0, y_0) > f(x_1, y_1)$.
- Chromosomen (Bitstrings) müssen zur Bestimmung ihrer Fitness zunächst dekodiert werden.



Ein *Simple Genetic Algorithm* (SGA)

sga(p_c, p_m, f, n, t, m)
    P ← *initialize population*(m, n)*; //* P Multimenge
    while t do
        Q ← *selectParents(P,f);*
        Q ← *applyCrossover(Q,*p_c*);*
        P ← *applyMutation(*Q, p_m*);*
        update t;
    end
    return P;

Input

p_c: Wahrscheinlichkeit für Crossover
p_m: Wahrscheinlichkeit für Mutation
f: Fitnessfunktion (inklusive Dekodierung)
n: Populationsgröße
t: Terminationsbedingung
m: Benötigte Länge der Kodierung

Parent selection

Ziel ist die Auswahl derjenigen Lösungen, die sich fortpflanzen, also zur nächsten Generation beitragen dürfen. Hierbei sollen Lösungen mit hoher Fitness eine höhere Wahrscheinlichkeit haben, sich durchzusetzen, als solche mit niedriger.

Mögliche Umsetzung:

- Einbau in Algorithmus gemäß der Populationsgröße: Wähle zur Berechnung von Q sooft ein Element aus der Population P aus, wie P groß ist (also n mal). **Achtung:** P und Q sind Multimengen (Mehrfachauswahl eines Elements möglich)!
- Umsetzung der Wahl durch fitness proportional selection (auch roulette wheel selection): Jede Lösung wird mit einer Wahrscheinlichkeit gewählt, die dem Anteil ihrer Fitness an der Gesamtfitness entspricht.

Roulette wheel selection

- Gegeben die Population $P = \{v_1, \dots, v_n\}$ berechne ihre Gesamtfitness
$$F = \sum_{i=1}^{n} f(v_i),$$
wobei f die Fitnessfunktion ist.
- Berechne für jede Lösung v_i ihre Wahrscheinlichkeit $p_i = \frac{f(v_i)}{F}$.
- In jedem Durchlauf wird die Lösung v_i mit Wahrscheinlichkeit p_i gewählt. Einzelne Durchläufe sind unabhängig voneinander; insbesondere kann die gleiche Lösung mehrfach gewählt werden.
- Mögliche Implementierung:
  - Berechne für jede Lösung v_i ihre kumulative Wahrscheinlichkeit $q_i = \sum_{j=1}^{i} p_j$.
  - Erzeuge zufällig gleichverteilt eine reelle Zahl $r \in [0,1]$. Falls $r \leq q_1$, wähle v_1. Sonst wähle v_i, sodass $q_{i-1} < r \leq q_i$.

Beispiel Roulette wheel selection

Population P der Größe 5:

Chromosom	Dekodiert	Fitness	p_i	q_i
$v_1 = (1001101000000011111110100110111111)$	(6.0845, 5.6522)	26.0196	0.27	0.27
$v_2 = (1110001001001101110010101000110010)$	(10.3484, 4.3803)	7.5800	0.08	0.35
$v_3 = (0000100000110010000001010111011101)$	(-2.5166, 4.3904)	19.5263	0.2	0.55
$v_4 = (1000110001011010011110000001110010)$	(5.2786, 5.5934)	17.4067	0.18	0.73
$v_5 = (0001110110010100110101111111000101)$	(-1.2552, 4.7345)	25.3412	0.26	0.99
		Σ 95.8738		

Fünfmaliges Generieren einer reellen Zufallszahl zwischen 0 und 1 könnte etwa $0.22; 0.86; 0.84; 0.21; 0.87$ ergeben. Dann besteht Q aus

$$Q = \{v_1, v_5, v_5, v_1, v_5\}.$$

Voraussetzungen Roulette wheel selection

Um Roulette wheel selection anwenden zu können, müssen die Fitness und der Suchraum die folgenden Eigenschaften haben:

- Es muss tatsächlich eine Fitnessfunktion f geben, Fitness also absolut messbar sein (und nicht nur relativ).
- Die Fitnessfunktion darf keine negativen Werte annehmen, also $f: S \to [0, \infty)$, wobei S der Suchraum ist (wir fassen zur Vereinfachung Dekodierung und Auswertung der Fitness in einen Schritt zusammen).
- Das Problem muss ein Maximierungsproblem sein, eine hohe Fitness also erstrebenswert.

Problemtransformation

Sind ein Suchraum S und eine beliebige Fitnessfunktion $f: S \to \mathbb{R}$ gegeben, lässt sich ein Optimierungsproblem auf folgende Weise transformieren:

- Statt die Funktion f zu minimieren, kann man äquivalent die Funktion $-f$ maximieren.
- Nimmt eine Fitnessfunktion f negative Werte an und man kennt eine Konstante $c > 0$, sodass $-c \leq f(s)$ für alle $s \in S$, so gilt $f': S \to [0, \infty)$ für $f' = f + c$.

Ein *Simple Genetic Algorithm* (SGA)

sga(p_c, p_m, f, n, t, m)
  $P \leftarrow$ *initialize population*(m, n); // P Multimenge
  while t do
    $Q \leftarrow$ *selectParents(P,f);*
    $Q \leftarrow$ *applyCrossover(Q,p_c);*
    $P \leftarrow$ *applyMutation(Q, p_m);*
    update t;
  end
  return P;

Input

p_c: Wahrscheinlichkeit für Crossover
p_m: Wahrscheinlichkeit für Mutation
f: Fitnessfunktion (inklusive Dekodierung)
n: Populationsgröße
t: Terminationsbedingung
m: Benötigte Länge der Kodierung

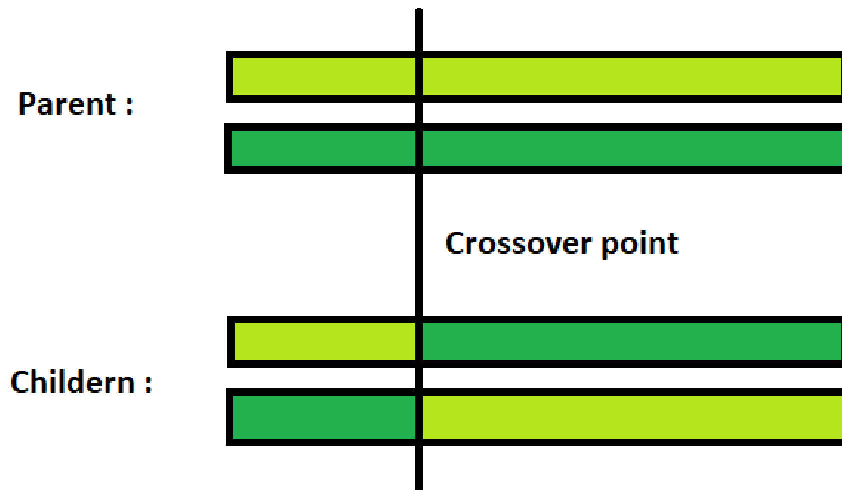
Crossover

Ziel ist das Mischen genetischer Informationen zweier Lösungen in einer neuen Lösung. Jede während der parent selection ausgewählte Lösung nimmt mit Wahrscheinlichkeit p_c an einem Crossover teil.

Mögliche Umsetzung:

- Bestimme für jedes Element v aus Q, ob es am Crossover teilnimmt. Erzeuge dazu beispielsweise zufällig gleichverteilt eine reelle Zahl $r \in [0,1]$. Falls $r \leq p_c$, ist c für den Crossover ausgewählt.
- Bilde aus den ausgewählten Elementen auf zufällige Weise Paare.
- Wende auf jedes Paar 1-Punkt Crossover an; hierbei entstehen zwei neue Lösungen (genannt Offspring oder Kindgenome).
- In Q werden die Chromosomen, die am Offspring teilgenommen haben, durch ihre Kindgenome ersetzt.

1-Punkt Crossover



https://www.geeksforgeeks.org/crossover-in-genetic-algorithm/

- Für Strings der Länge m bestimme eine zufällige natürliche Zahl $1 \leq p \leq m - 1$.
- Zerschneide die beiden Elternstrings jeweils hinter der Position p.
- Die Kinder entstehen durch kreuzweises Rekombinieren der Eltern.

Beispiel Crossover

Bei einer Crossover-Wahrscheinlichkeit $p_c = 0.25$ und einer Populationsgröße von 5 erwarten wir, dass meistens ein Element für Crossover ausgewählt wird.

Für die Population $Q = \{v_1, v_5, v_5, v_1, v_5\}$ von oben generiere fünf reelle Zufallszahlen zwischen 0 und 1 (unabhängig und gleichverteilt):

$$0.67;\ 0.93;\ 0.89;\ 0.06;\ 0.12$$

Damit nehmen v_1 und v_5 am Crossover teil (als viertes und fünftes Element der Population).

Beispiel 1-Punkt Crossover

Wende 1-Punkt Crossover an auf

$$v_1 = (100110100000001111111010011011111)$$
$$v_5 = (000111011001010011010111111000101)$$

- Generiere zufällige natürliche Zahl zwischen 1 und 32. Ergebnis z.B. 25.
- Teile v_1 und v_5 nach Position 25 auf:

$$v_1 = (1001101000000011111110100 \mid 11011111)$$
$$v_5 = (0001110110010100110101111 \mid 11000101)$$

- Berechne Kinder durch kreuzweise Rekombination:

$$o_1 = (100110100000001111111010011000101)$$
$$o_2 = (000111011001010011010111111011111)$$

- Aktualisiere die Population Q entsprechend. In unserem Beispiel gilt jetzt

$$Q = \{v_1, v_5, v_5, o_1, o_2\}.$$

Ein *Simple Genetic Algorithm* (SGA)

sga(p_c, p_m, f, n, t, m)
  P ← *initialize population*(m, n); // P Multimenge
  while t do
    Q ← *selectParents(P,f);*
    Q ← *applyCrossover(Q,*p_c*);*
    P ← *applyMutation(*Q, p_m*);*
    update t;
  end
  return P;

Input

p_c: Wahrscheinlichkeit für Crossover
p_m: Wahrscheinlichkeit für Mutation
f: Fitnessfunktion (inklusive Dekodierung)
n: Populationsgröße
t: Terminationsbedingung
m: Benötigte Länge der Kodierung

Mutation

Ziel ist das zufällige, lokale Ändern von Chromosomen. Jedes zu mutierende Chromosom wird durchlaufen und jedes Gen wird mit Wahrscheinlichkeit p_m mutiert.

Mögliche Umsetzung:

- Durchlaufe Q Chromosom für Chromosom und dabei jedes Chromosom Bit für Bit.
- Erzeuge für jedes Bit b zufällig gleichverteilt eine reelle Zahl $r \in [0,1]$. Falls $r \leq p_\mathrm{m}$, wird b geswitcht.
- Die entstehende Population bildet die Eingabe für die nächste Iteration des SGA.

UNIKASSEL VERSITÄT

SOFTWARE ENGINEERING UNIVERSITÄT KASSEL

Beispiel Mutation

Eine häufig gewählte Mutationswahrscheinlichkeit ist $p_{\mathrm{m}} = {}^{1}/_{m}$, wo m die Länge der Bitstrings bezeichnet. In unserem Beispiel: ${}^{1}/_{m} \approx 0.03$

Q (Input-Population)	Generierte Zufallszahlen	P (Output-Population)
$v_1 = (100110100000001111111010011011111)$	Einmal unter 0.03 (an Pos 5)	100100100000001111111010011011111
$v_5 = (000111011001010011010111111000101)$	Nie unter 0.03	000111011001010011010111111000101
$v_5 = (000111011001010011010111111000101)$	Nie unter 0.03	000111011001010011010111111000101
$o_1 = (100110100000001111111010011000101)$	Zweimal unter 0.03 (Pos 11 und 17)	100110100010001101111010011000101
$o_2 = (000111011001010011010111111011111)$	Einmal unter 0.03 (an Pos 31)	000111011001010011010111111011011

Ein *Simple Genetic Algorithm* (SGA)

sga(p_c, p_m, f, n, t, m)
  $P \leftarrow$ *initialize population*(m, n)*; //* P Multimenge
  while t do
    $Q \leftarrow$ *selectParents(P,f);*
    $Q \leftarrow$ *applyCrossover(Q,*p_c*);*
    $P \leftarrow$ *applyMutation(*Q, p_m*);*
    update t;
  end
  return P;

Input

p_c: Wahrscheinlichkeit für Crossover
p_m: Wahrscheinlichkeit für Mutation
f: Fitnessfunktion (inklusive Dekodierung)
n: Populationsgröße
t: Terminationsbedingung
m: Benötigte Länge der Kodierung

Terminationsbedingung

Aufgabe der Terminationsbedingung ist es, die evolutionäre Berechnung irgendwann zu stoppen.

Mögliche Umsetzung:

- Zähle die Anzahl der Iterationen.
- Füge dem SGA einen Parameter hinzu, der die Anzahl der gewünschten Iterationen angibt. Die Berechnung stoppt, wenn diese Anzahl erreicht ist.

Zusammenfassung

Der Simple Genetic Algorithm (SGA)

- Benutzt Bitstrings als Kodierung
- Evolviert eine Population von Elementen (und nicht ein einzelnes Element)
- Selektion ist randomisiert und bevorzugt fitte Chromosomen
- Mutation und Crossover werden als Suchoperatoren eingesetzt
  - Auch beide randomisiert angewendet
  - Crossover: Kreuzweises kombinieren von Teilen existierender Lösungen
  - Mutation: Bitswitch

Ausblick

- Welche anderen Möglichkeiten gibt es, Lösungen zu repräsentieren? Wie definieren wir auf diesen Mutation und Crossover?
- Gibt es sinnvollere Terminationsbedingungen als die Anzahl der Iterationen?
- Ist Selektion anhand der absoluten Fitness immer angemessen?
- Sollte auch Wettbewerb zwischen der Eltern- und der Kindgeneration stattfinden?